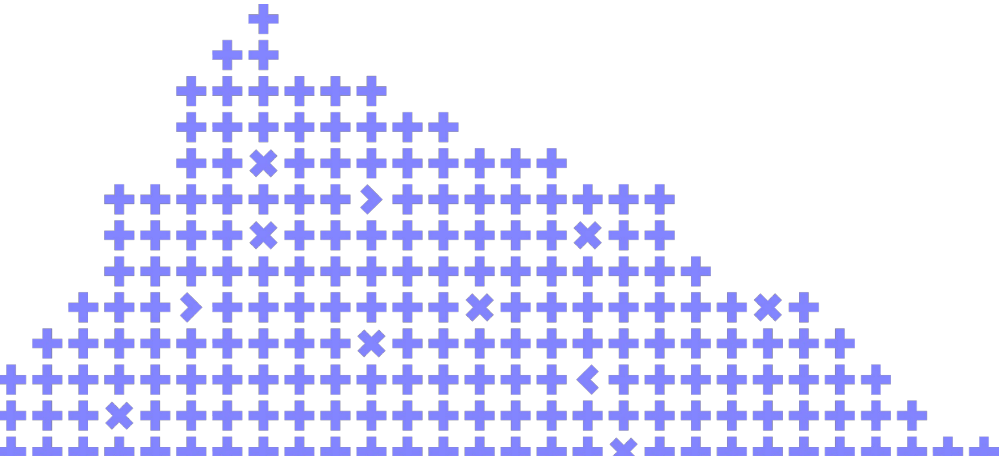
How Did We Build Rebalance into the Distributed Database Architecture

Vladislav Pyatkov



++ High Load Armenia

Co-organizer

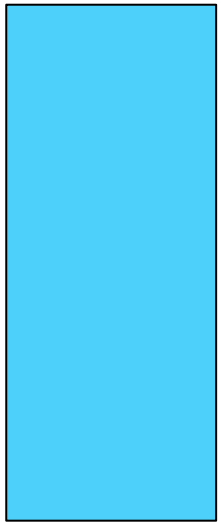
Yandex

About me

- Vladislav Pyatkov
- I work at GridGain
- Our product is based on Apache Ignite
- Today I am an Apache Ignite committer
- I have been developing distributed databases for more than five years

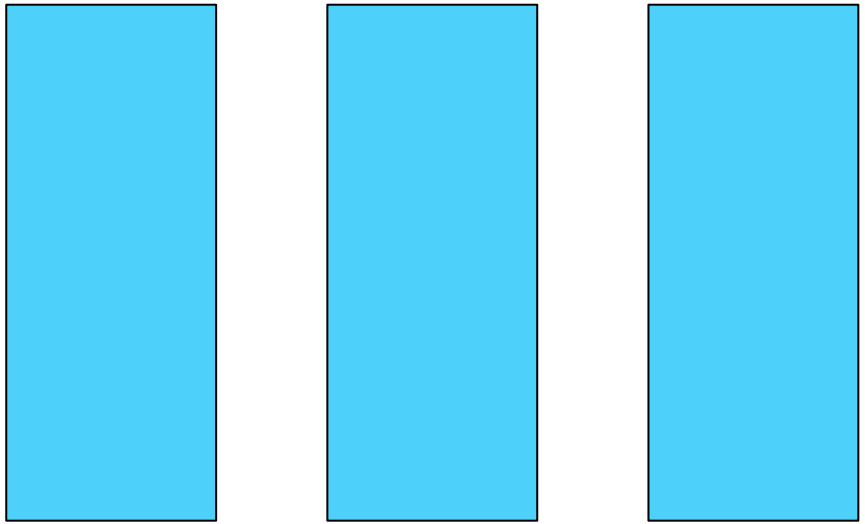
Node

One server in the distributed database



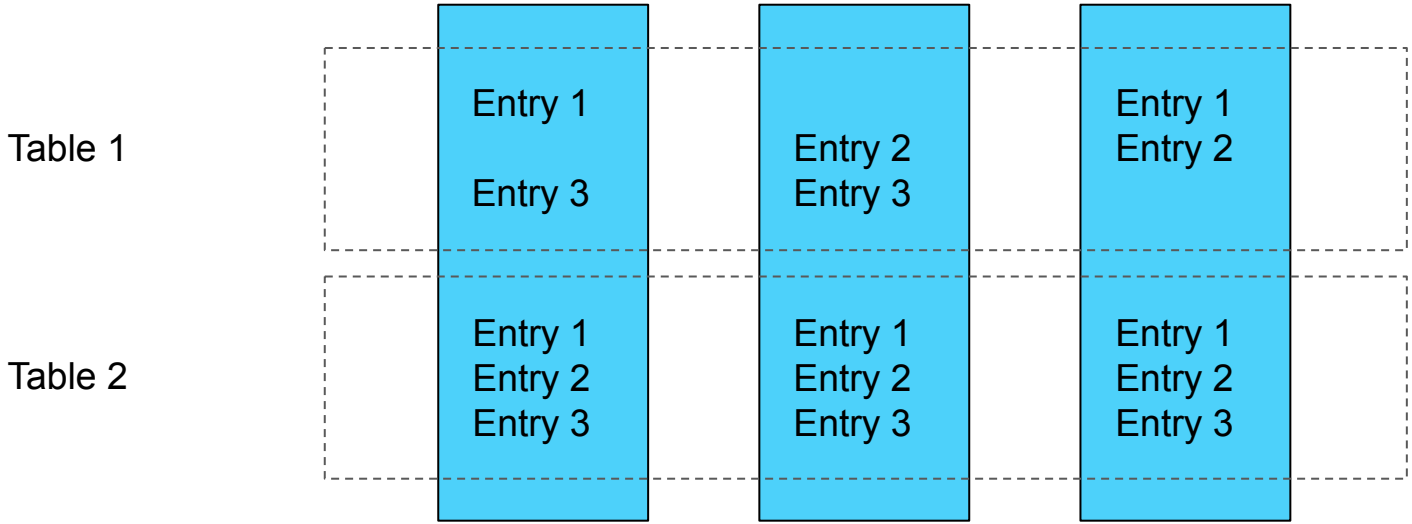
Cluster

A set of nodes is also called topology



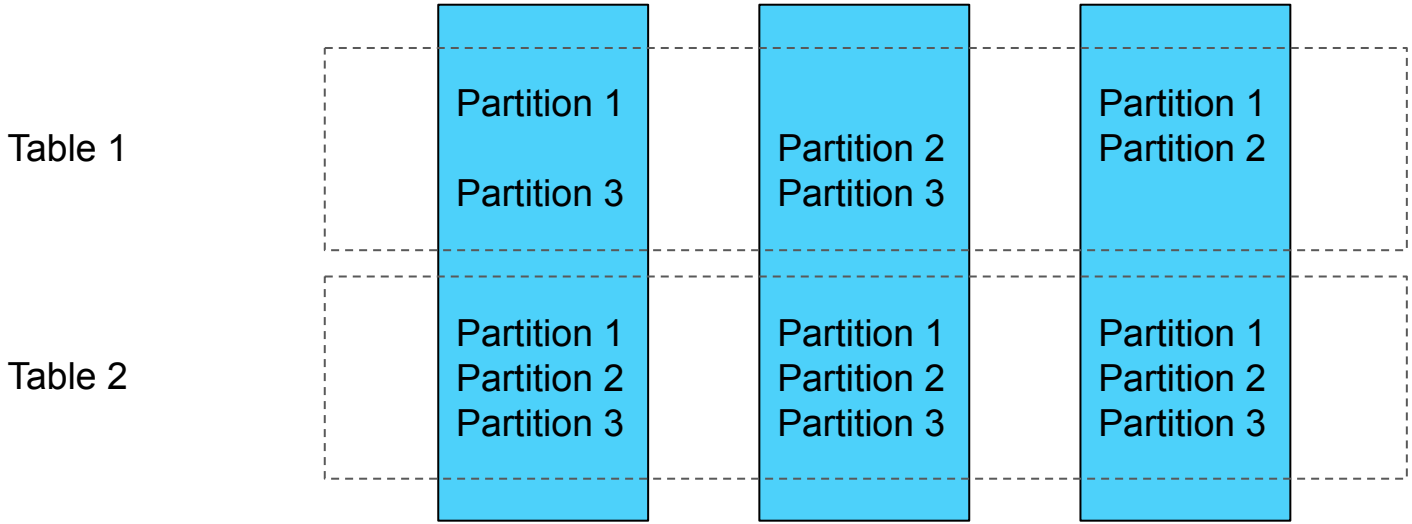
Data

The rule to map entries to nodes is called affinity function



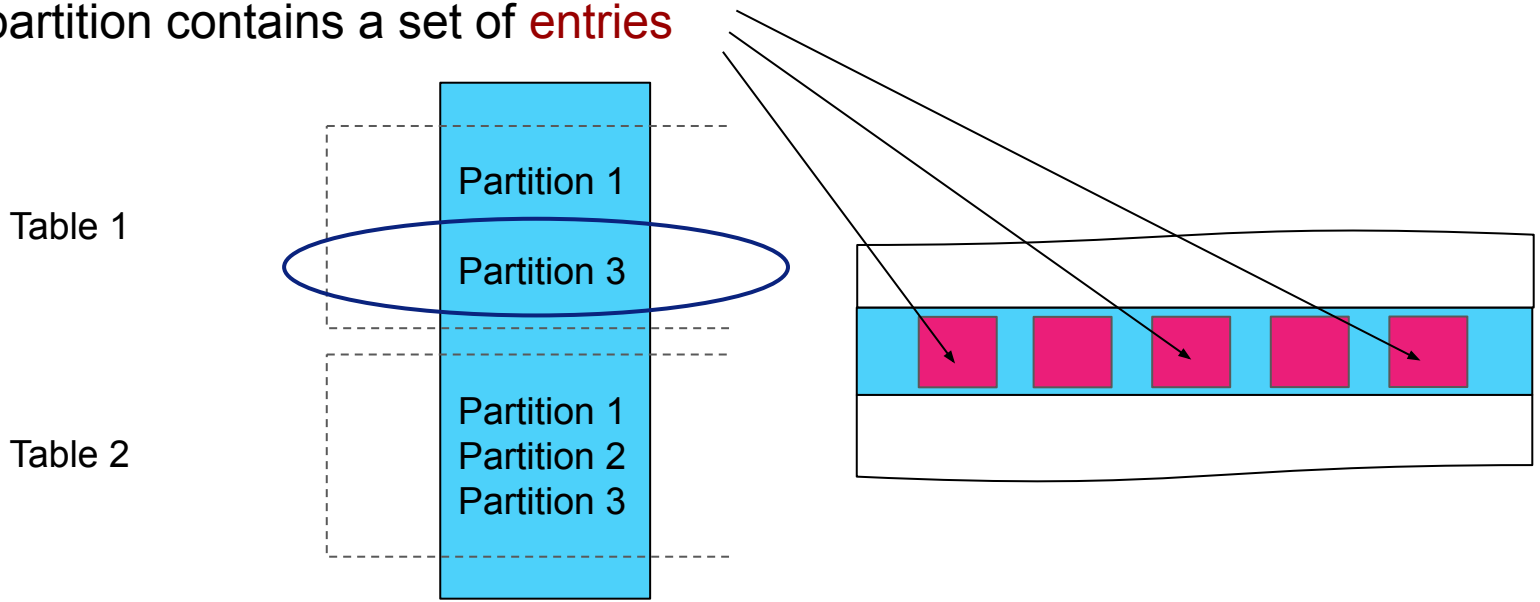
Partition

The result of the affinity function calculation is called distribution



Partition internals

Each partition contains a set of entries



Partition

Copies of one partition on different nodes are called replicas

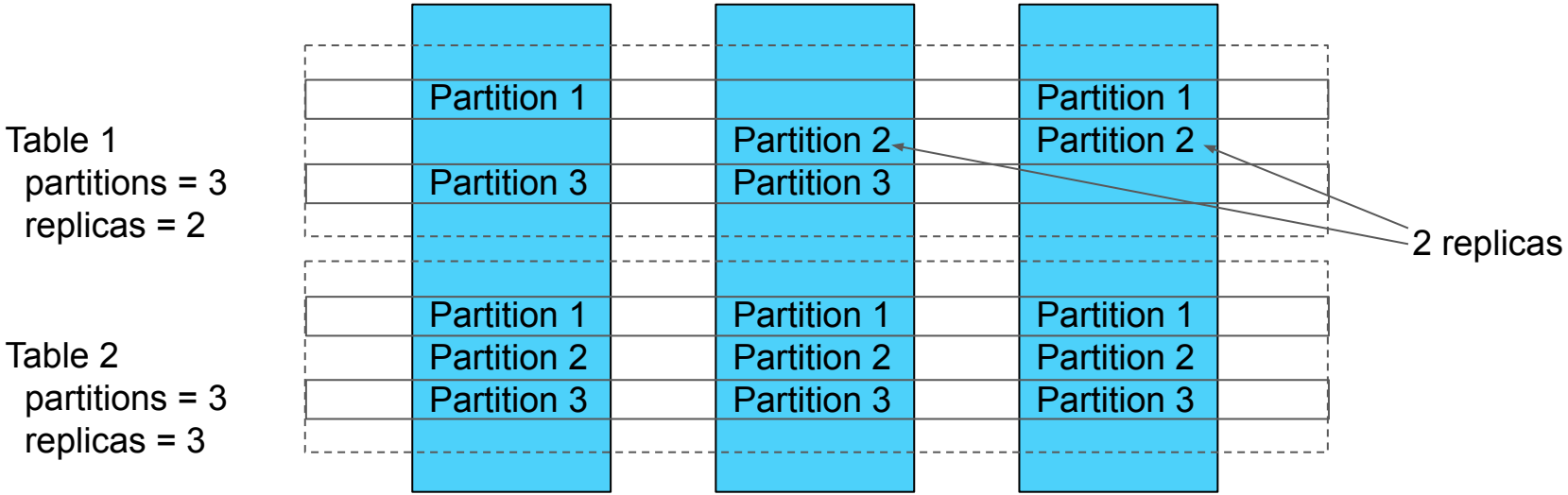


Table of content

- ➜ Data movement processes
  - Rebalancing
  - Replication

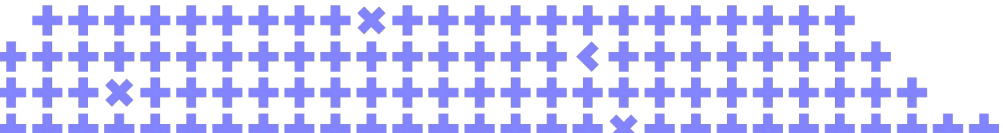


Table of content

- Data movement processes
  - Rebalancing
  - Replication
- ➜ Replication in a stable distribution
  - Rebalancing does not happen.
  - Replication happens.

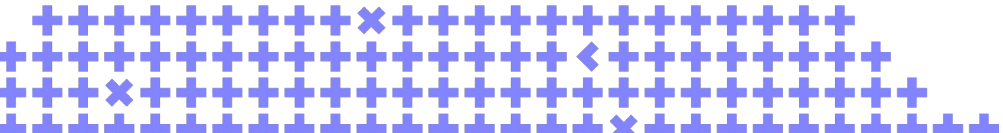


Table of content

- Data movement processes
  - Rebalancing
  - Replication
- Replication in a stable distribution
  - Rebalancing does not happen.
  - Replication happens.
- ➜ Changing partition distribution
  - Topology changes due to adding and removing nodes.
  - Replication factor changes.
  - Other scenarios when partition distribution is changed.

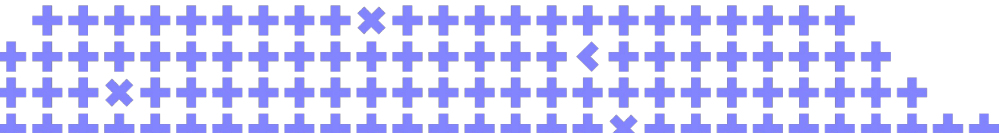


Table of content

- Data movement processes
  - Rebalancing
  - Replication
- Replication in a stable distribution
  - Rebalancing does not happen.
  - Replication happens.
- Changing partition distribution
  - Topology changes due to adding and removing nodes.
  - Replication factor changes.
  - Other scenarios when partition distribution is changed.
- ➜ What happens if all replicas should be moved to new nodes during redistribution?
  - RAFT consensus

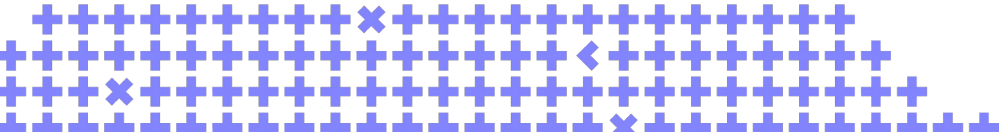
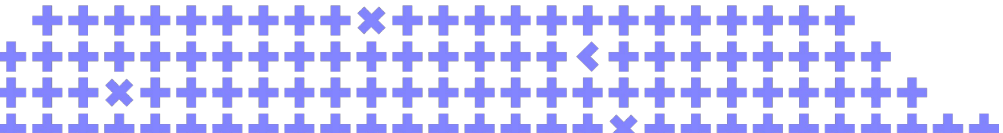
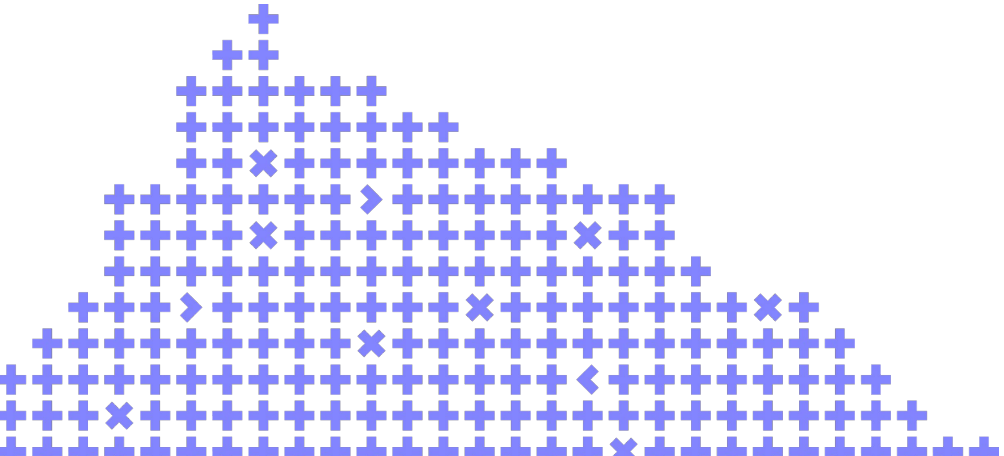


Table of content

- Data movement processes
  - Rebalancing
  - Replication
- Replication in a stable distribution
  - Rebalancing does not happen.
  - Replication happens.
- Changing partition distribution
  - Topology changes due to adding and removing nodes.
  - Replication factor changes.
  - Other scenarios when partition distribution is changed.
- What happens if all replicas should be moved to new nodes during redistribution?
  - RAFT consensus
- ➜ Conclusion



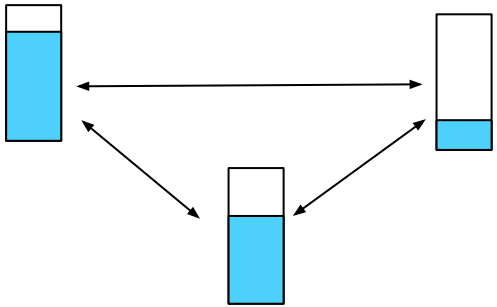
Data movement processes



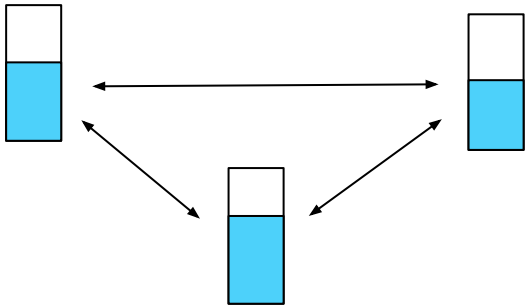
Rebalancing

A process of moving data between nodes in order to make the distribution as close to the uniform as possible

Before

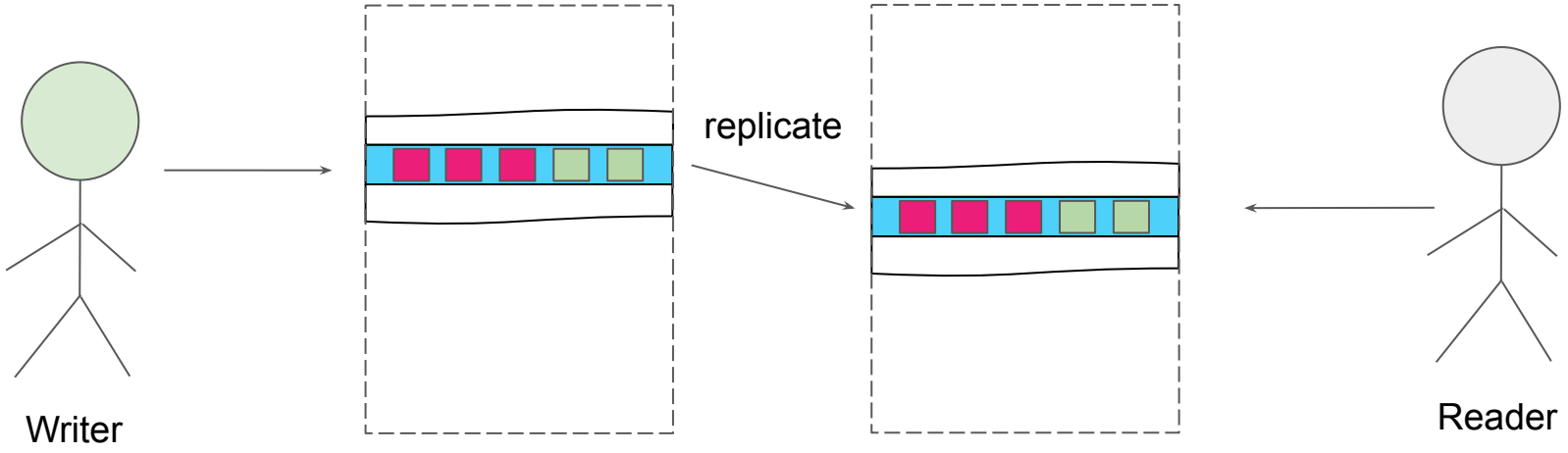


After



Replication

Ensure consistency across partition replicas

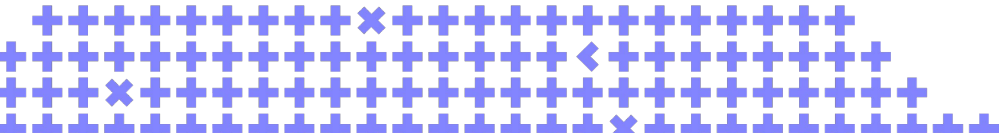


Two approaches to replication

Data entries are transferred between replicas.

Log records are transferred between replicas.

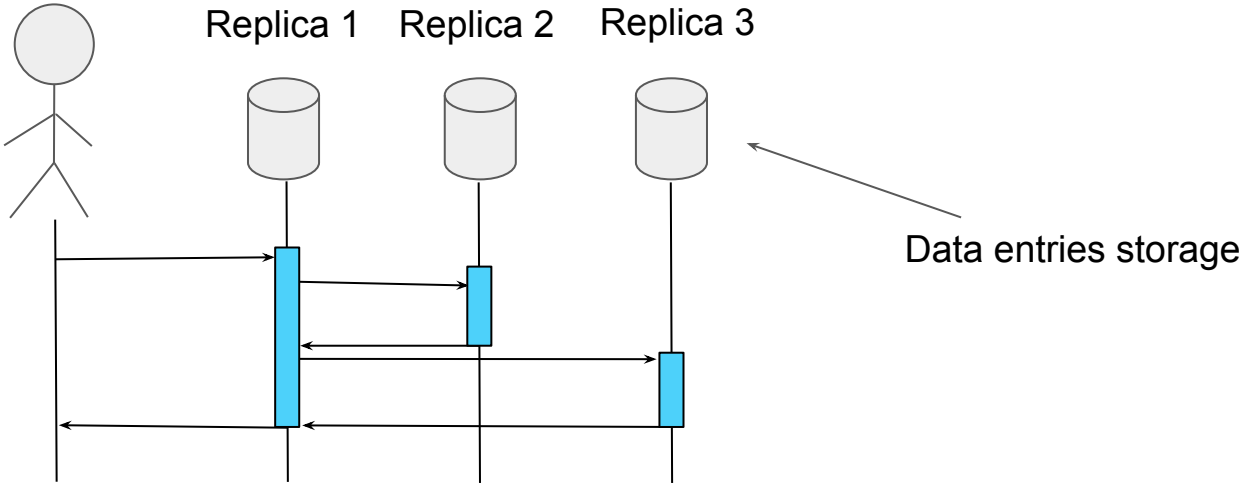
For both cases, we assume that the data (the partition) is replicated on three nodes.
The set of nodes is called the partition's replication group.



Two approaches to replication

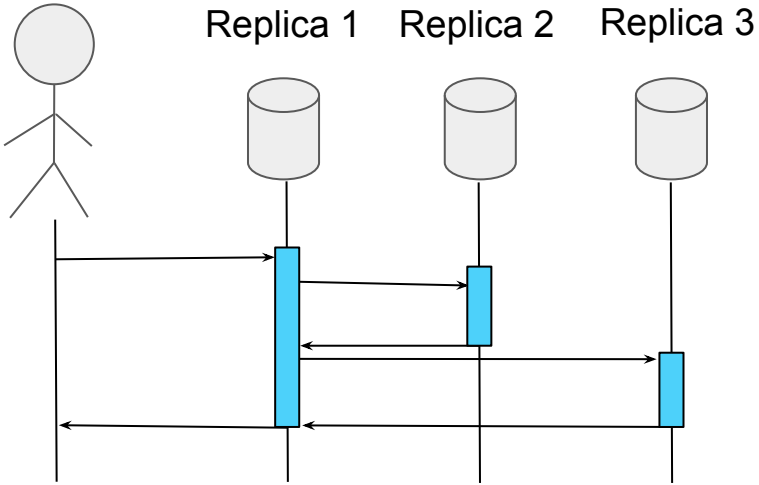
Data entries are moved between replicas.

Log records are transferred between replicas.

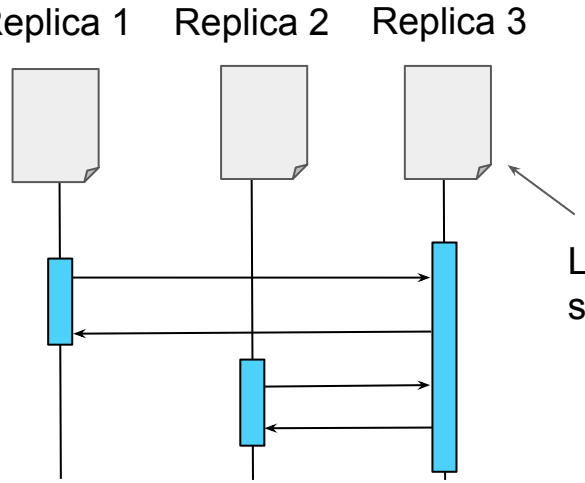


Two approaches to replication

Data entries are moved between replicas

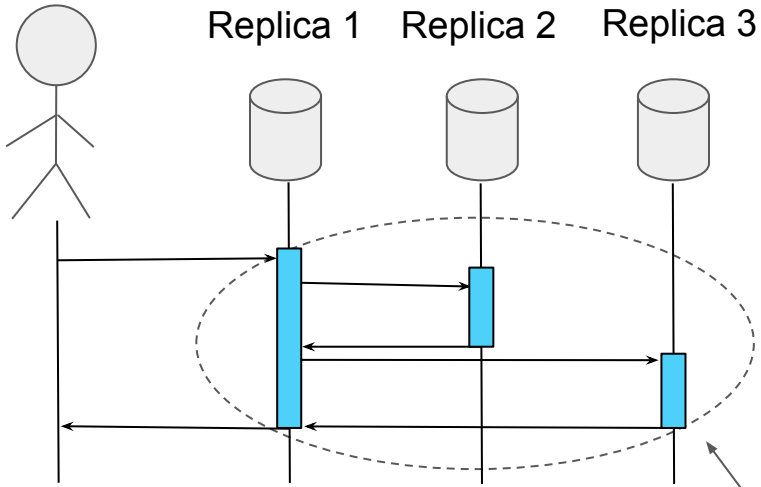


Log records are transferred between replicas

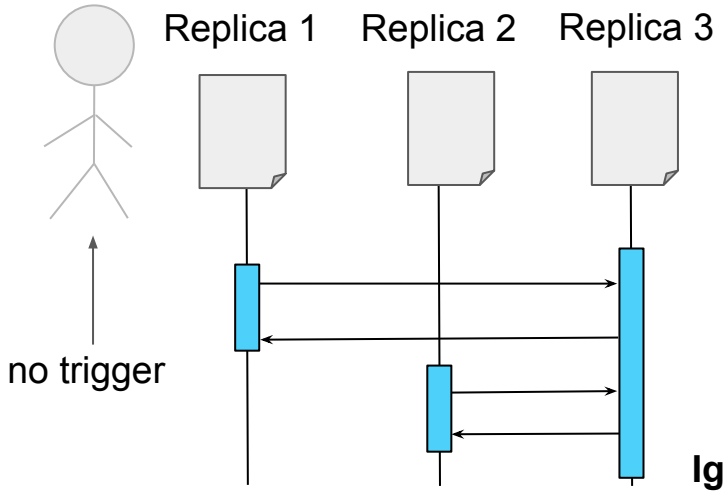


Two approaches to replication

Data entries are moved between replicas.



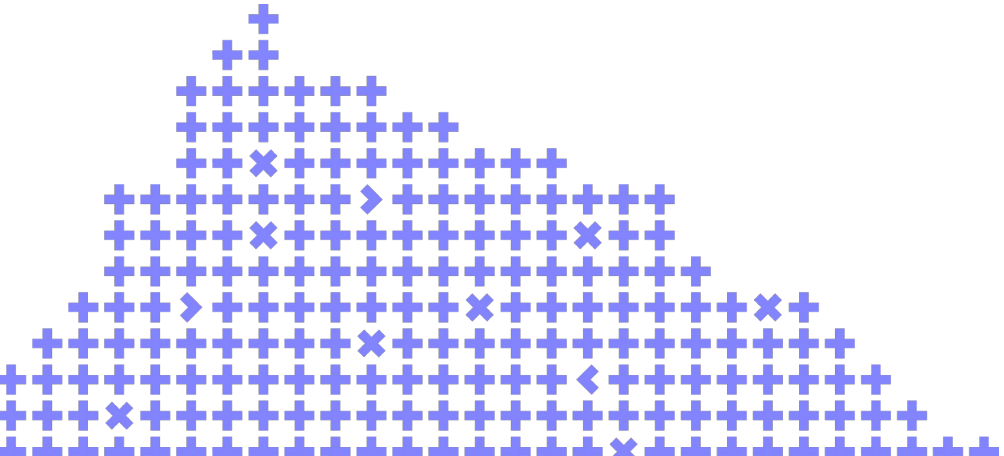
Log records are transferred between replicas.

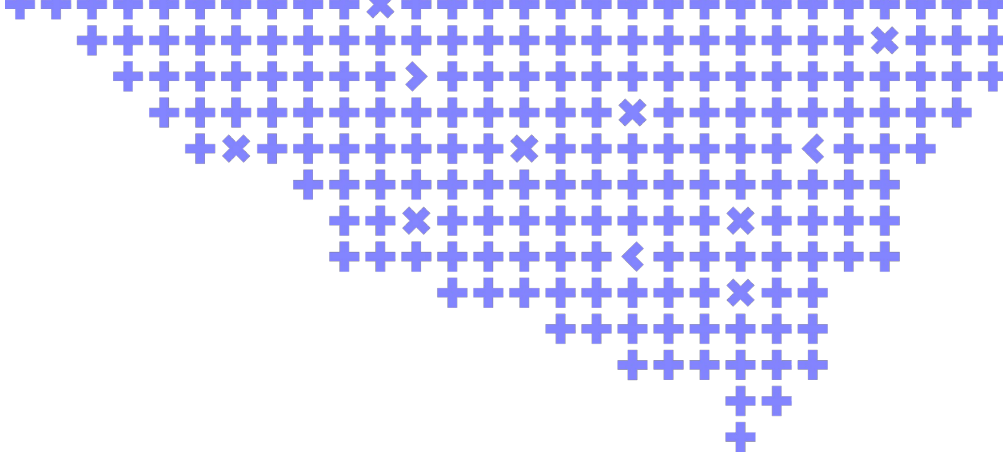


Ignite v3

Operation execution time

Replication in a stable distribution



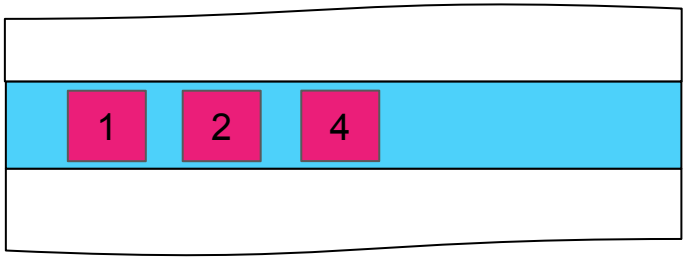


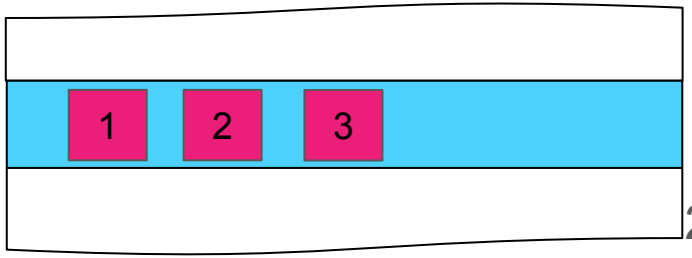
Partition lag is handled by replication

6 => insert(4)
5 => update(2)
4 => delete(3)
3 => insert(3)
2 => insert(2)
1 => insert(1)

{6, insert(4);
5, update(2);
4, delete(3)}

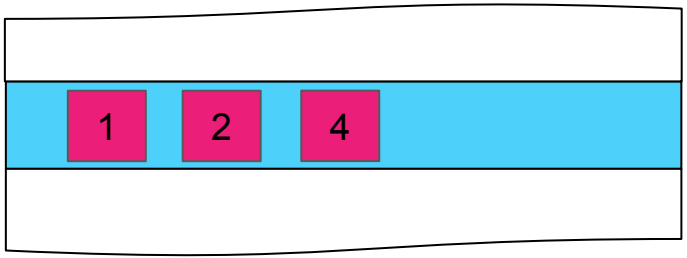
3 => insert(3)
2 => insert(2)
1 => insert(1)





Partition lag is handled by replication

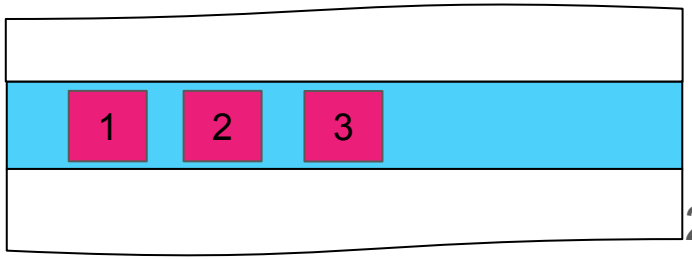
6 => insert(4)
5 => update(2)
4 => delete(3)
3 => insert(3)
2 => insert(2)
1 => insert(1)



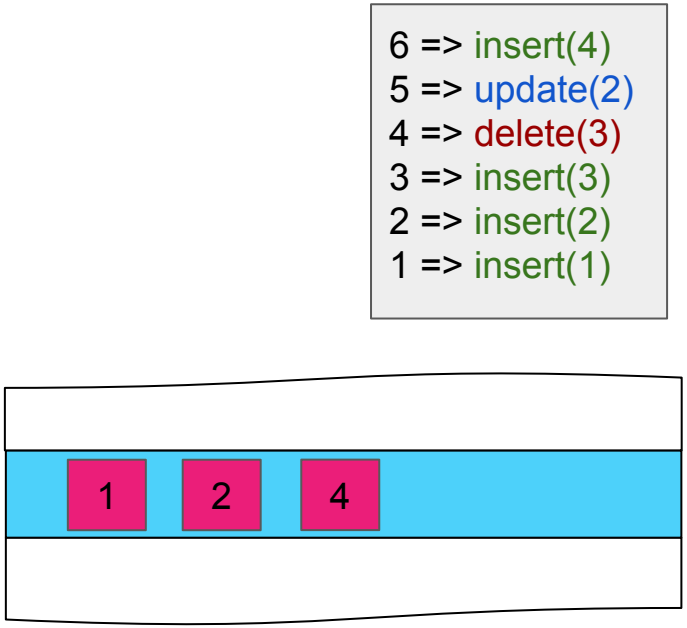
6 => insert(4)
5 => update(2)
4 => delete(3)

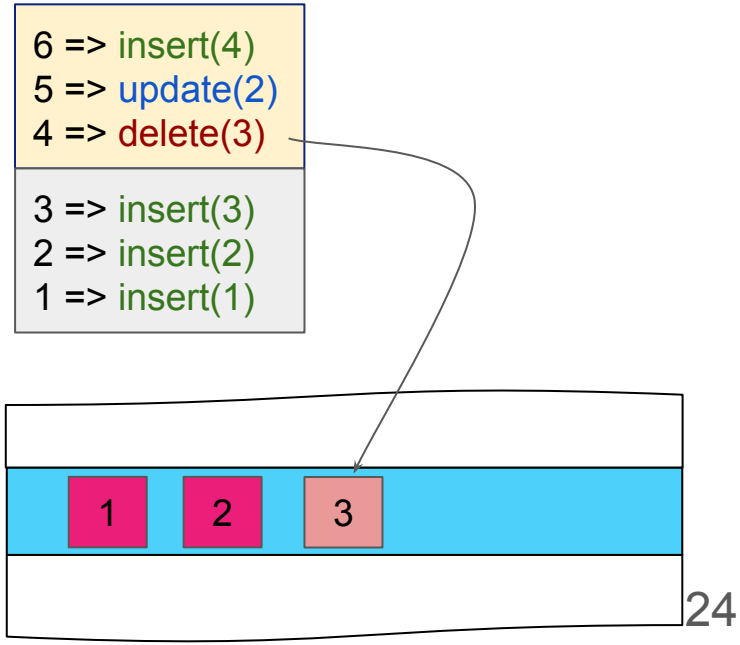
3 => insert(3)
2 => insert(2)
1 => insert(1)

These updates will be applied serially to the storage.

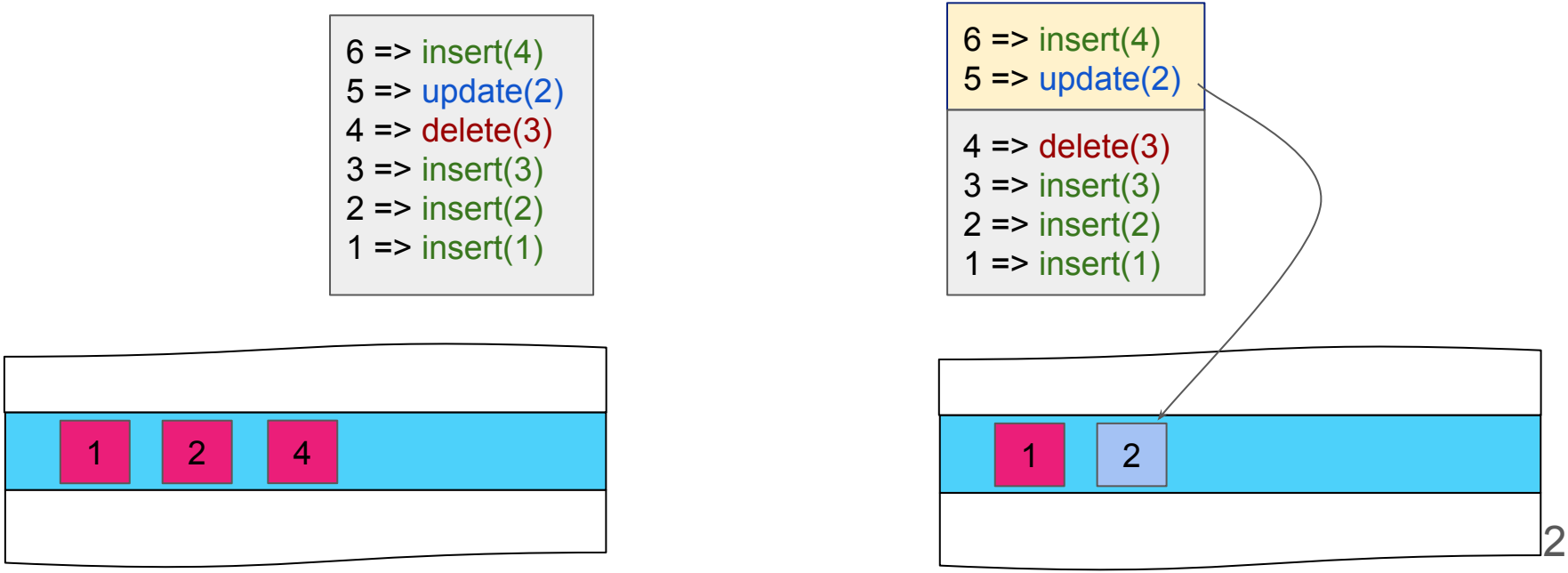


Partition lag is handled by replication

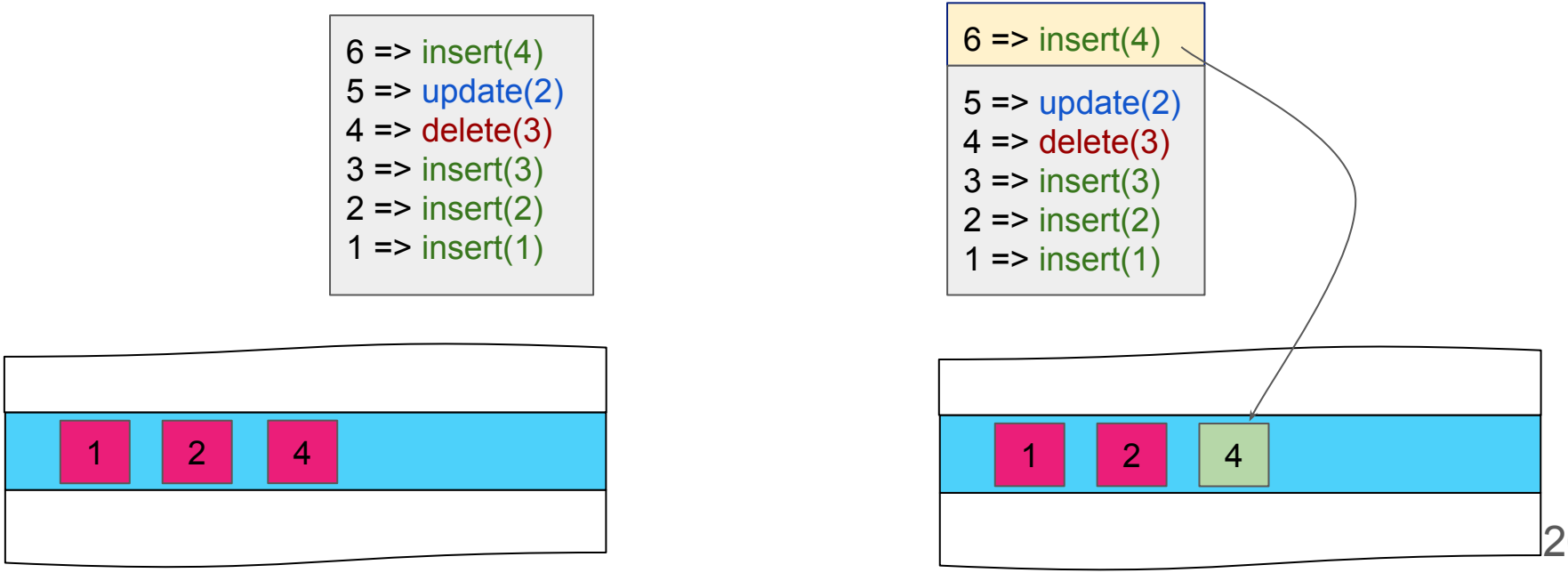




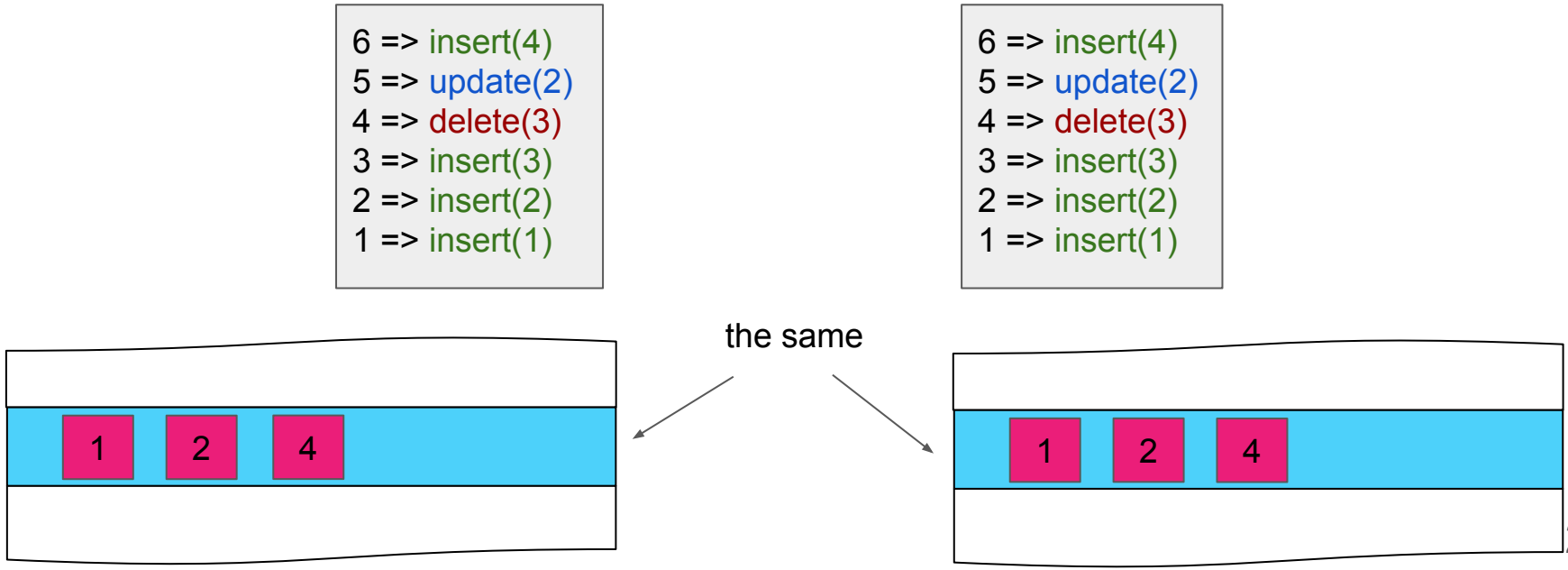
Partition lag is handled by replication



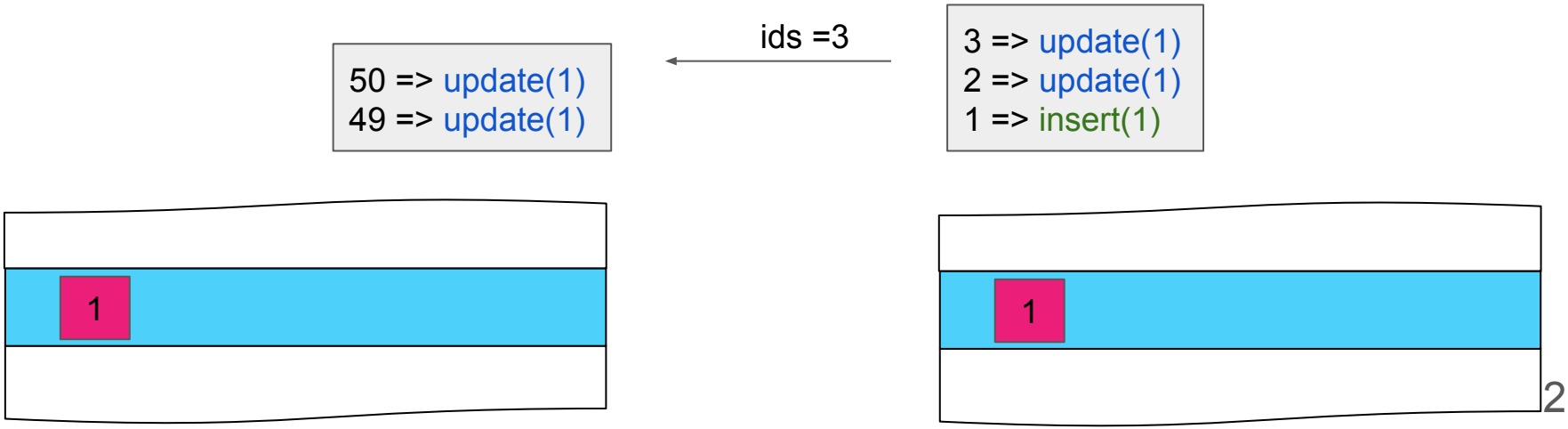
Partition lag is handled by replication



Partition lag is handled by replication



Truncated log



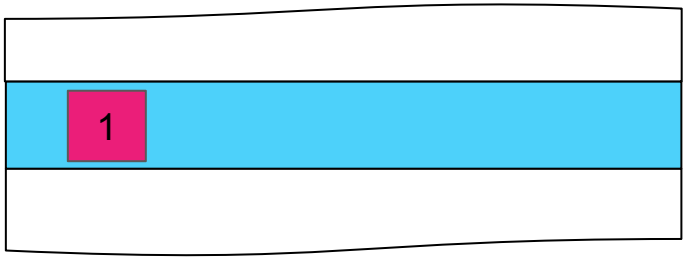
Truncated log

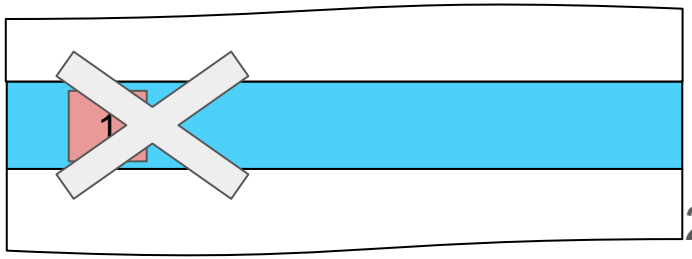
Clear the storage before applying a snapshot.

50 => update(1)
49 => update(1)

Snapshot

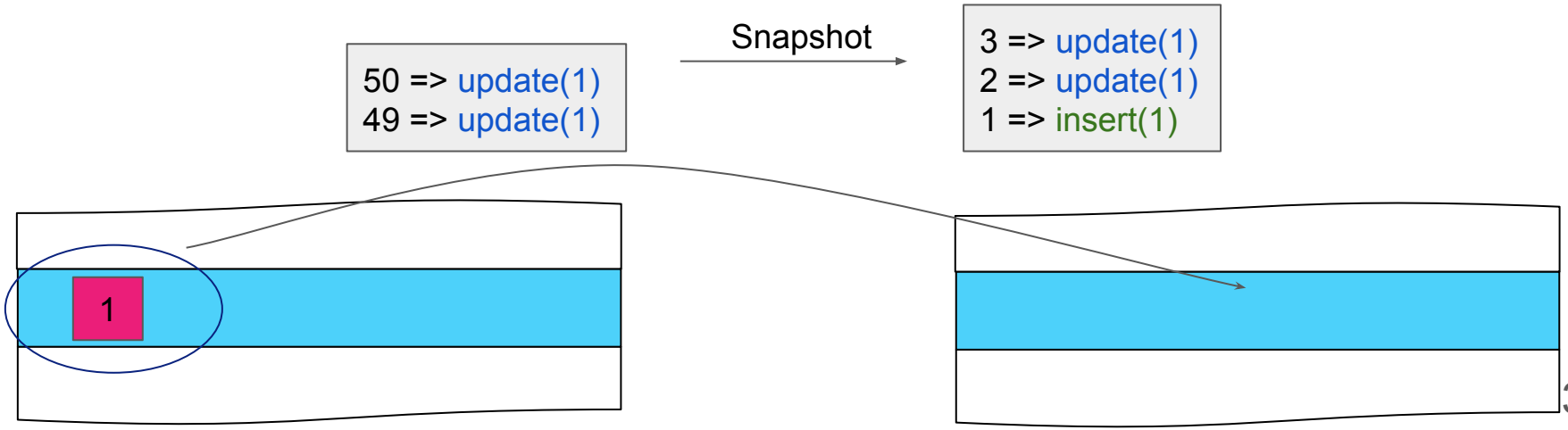
3 => update(1)
2 => update(1)
1 => insert(1)





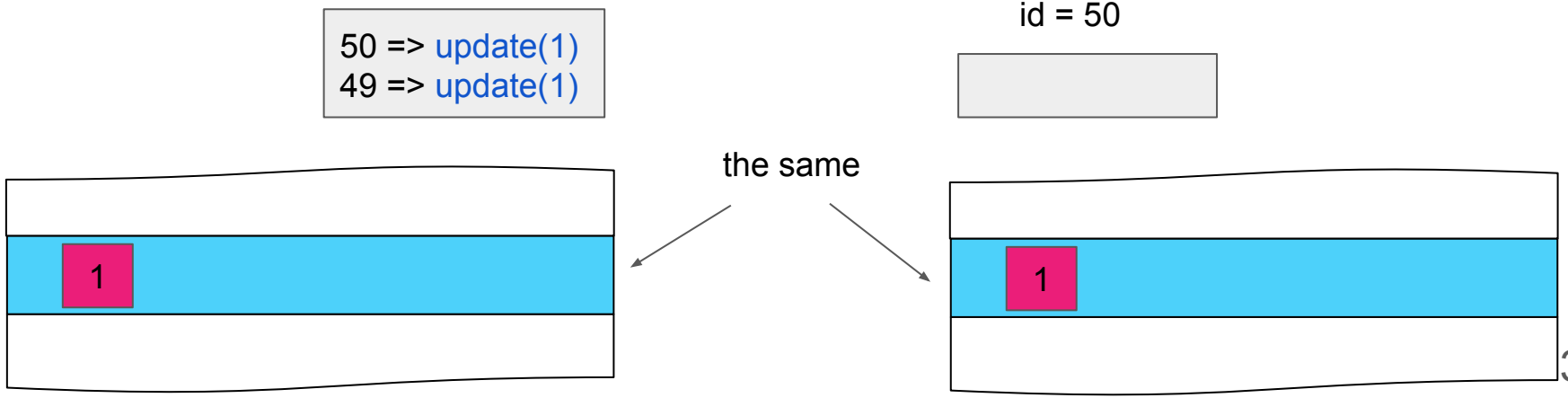
Truncated log

Write all data from the snapshot to the storage.

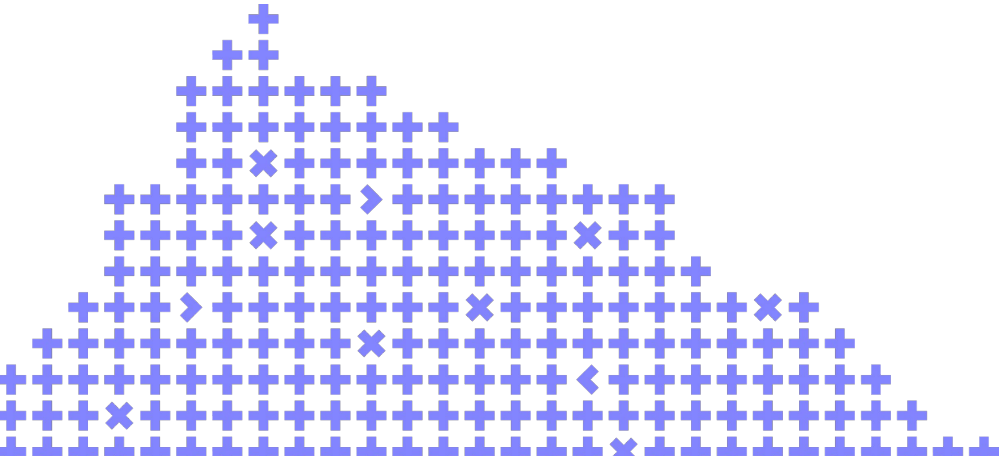


Truncated log

Although the logs are different, the persistent states are the same.

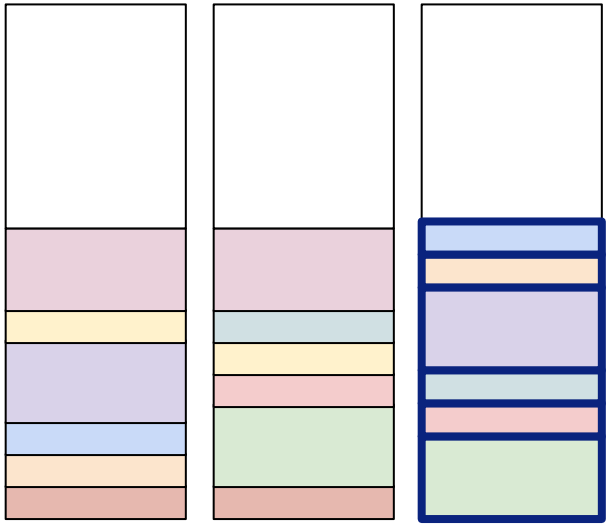


Changing partition distribution

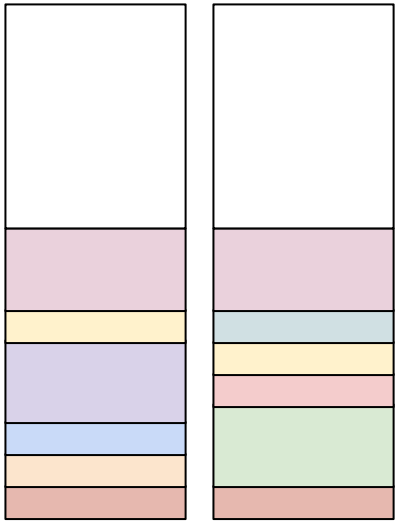


Removing a node

The replication factor is 2.



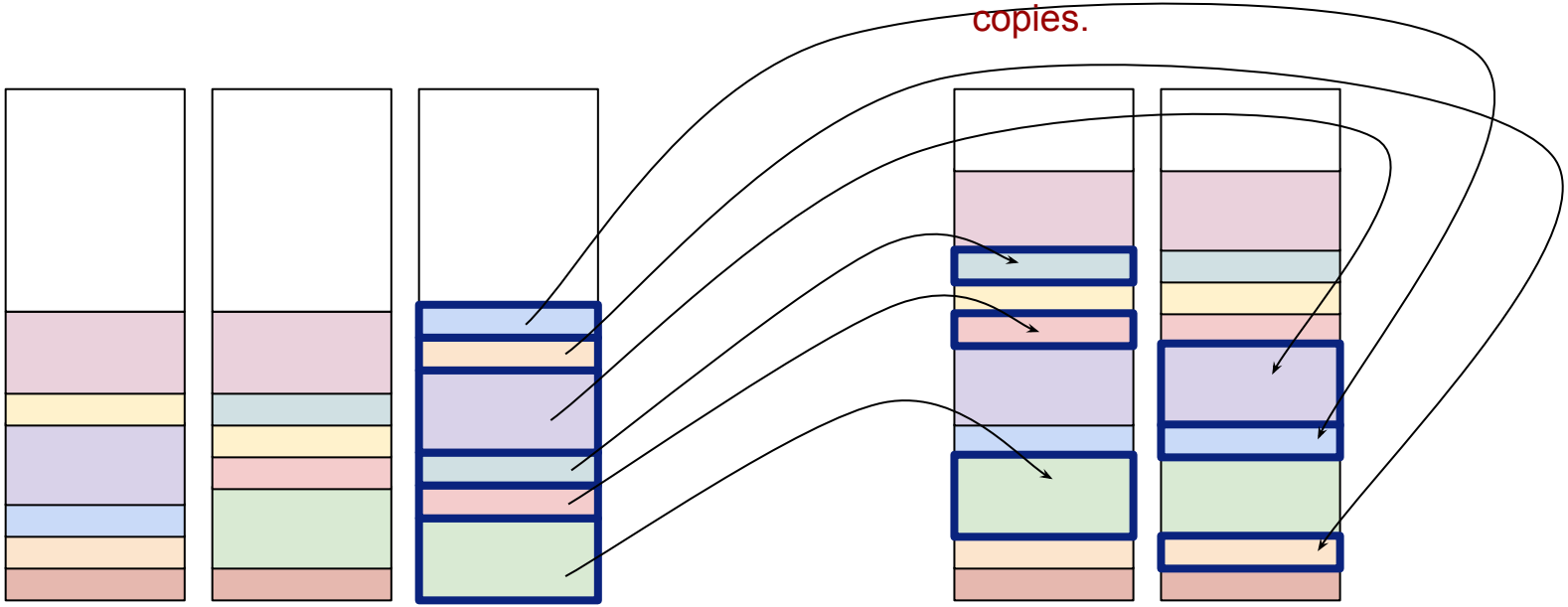
The replication factor is temporarily violated. Hence, the factor needs to be recovered.



Removing a node: moving to a new distribution

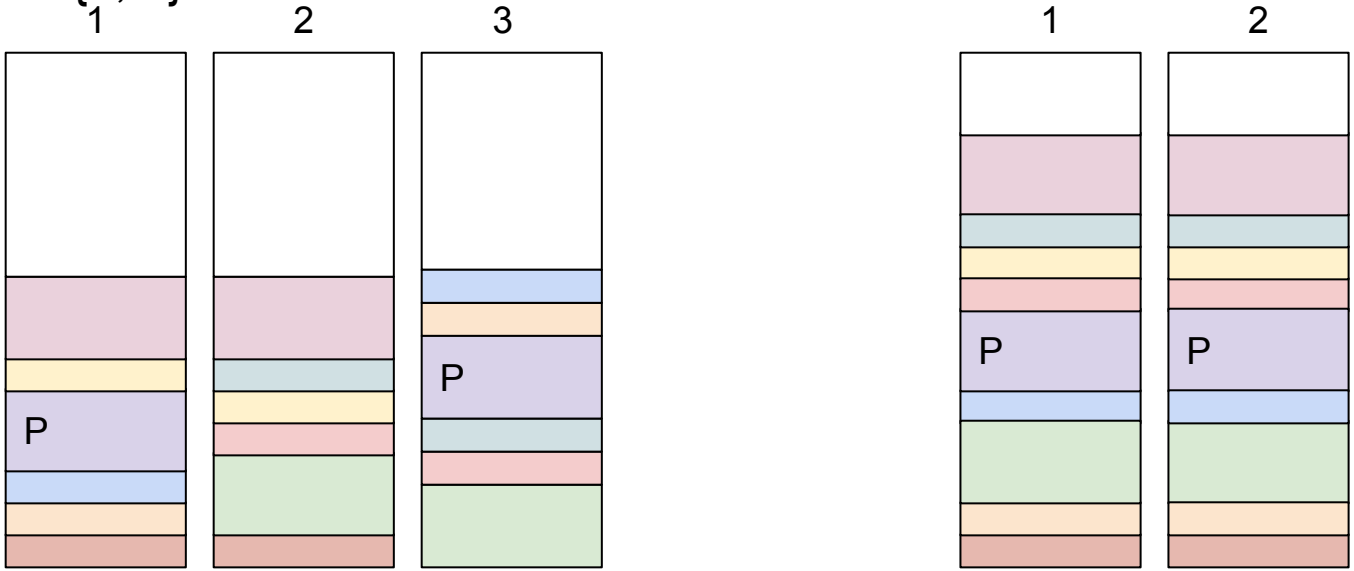
The replication factor is 2.

Again, each partition contains two copies.



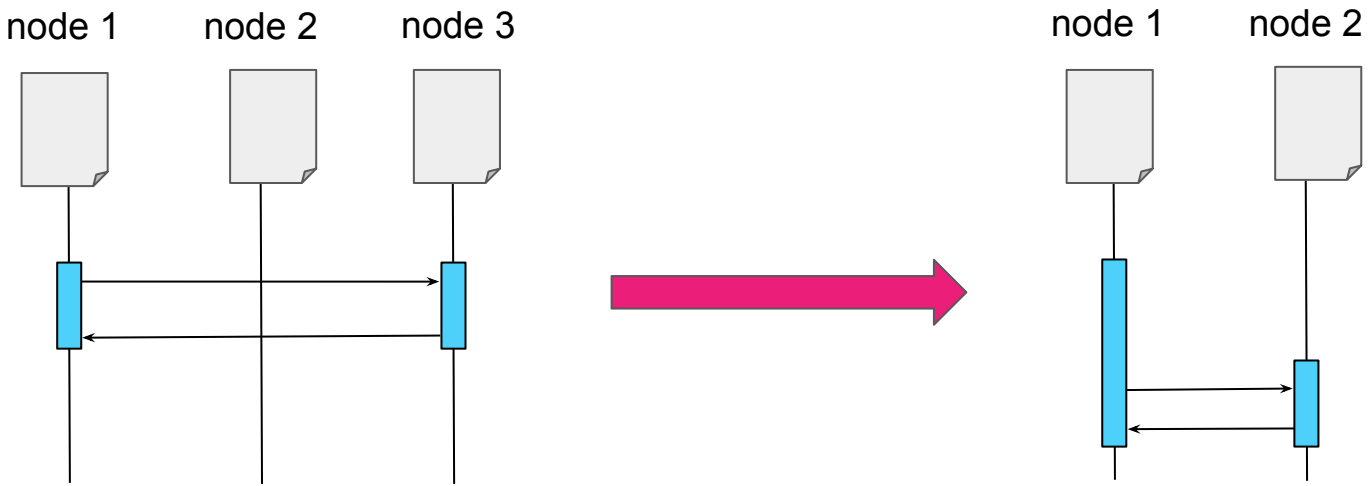
Removing a node: choose one partition

The replication group for partition <P> is changed from {1, 3} old distribution to {1, 2} new distribution.



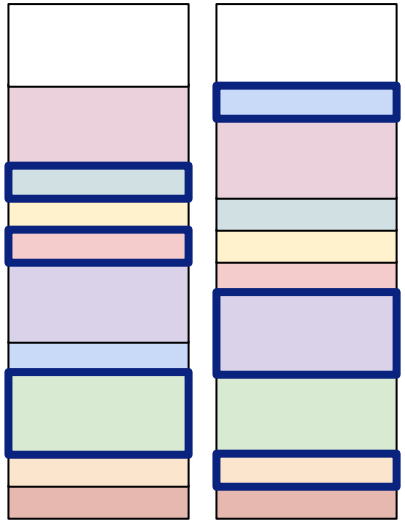
Removing a node: changing in replication groups

The replication group for partition <P> is changed from {1, 3} old distribution to {1, 2} new distribution.

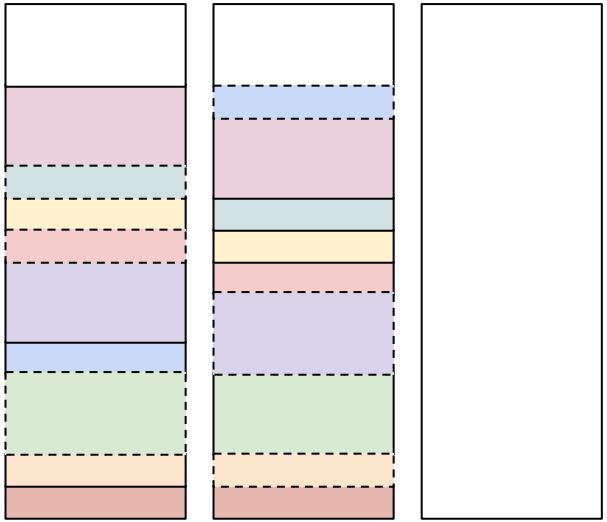


Adding nodes

Data is distributed across both cluster nodes.



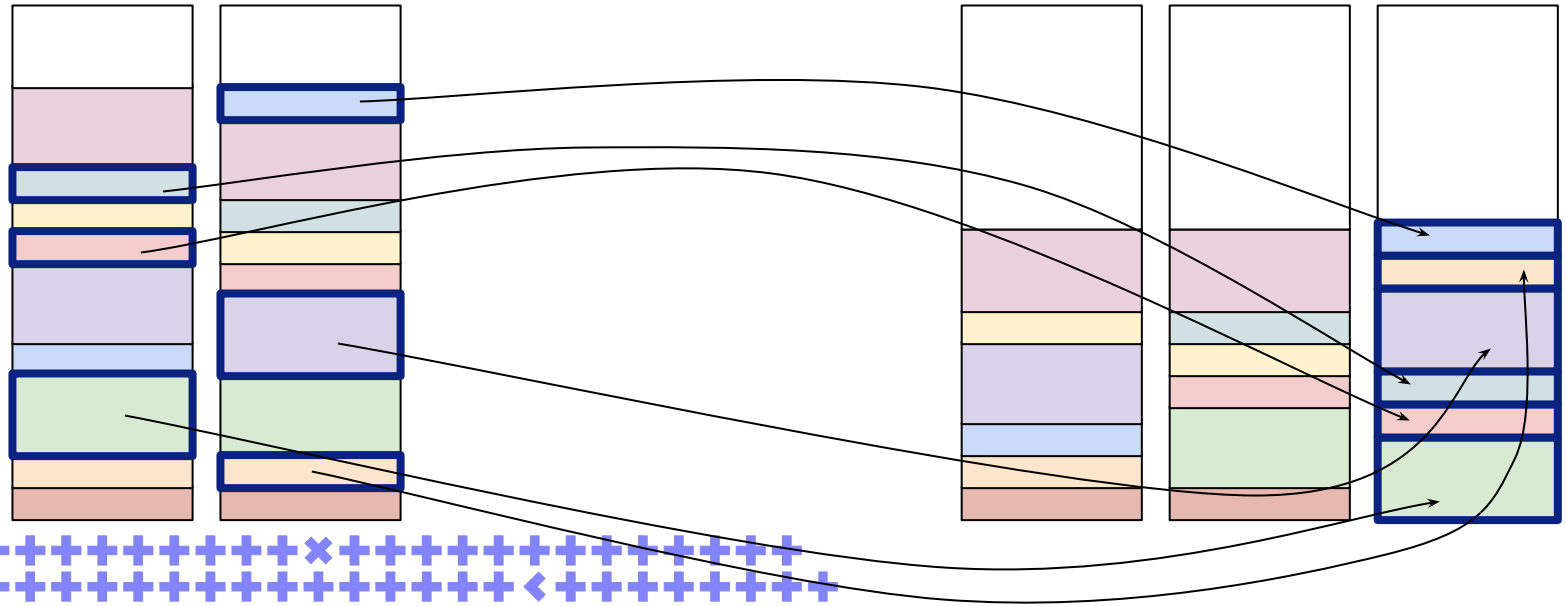
The third node space is empty. Hence, the space should be adjusted.



Adding nodes: moving to a new distribution

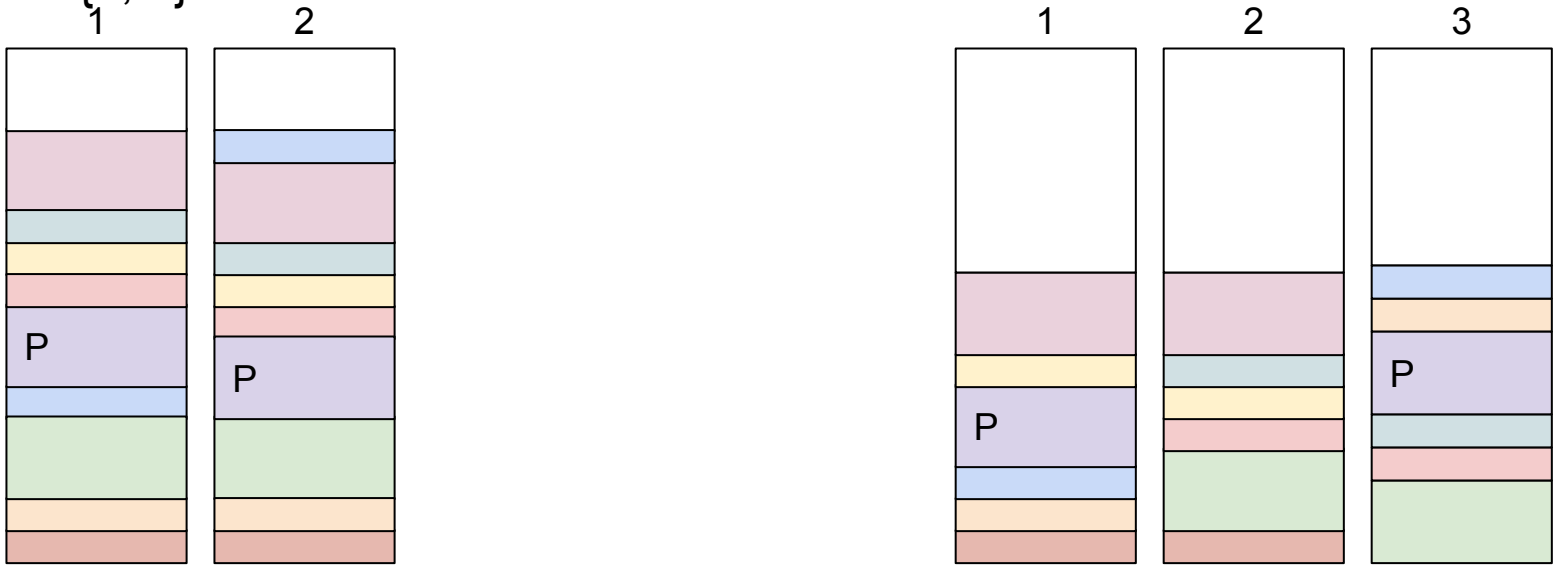
Data is distributed across both cluster nodes.

Data is distributed across three cluster nodes as evenly as possible.



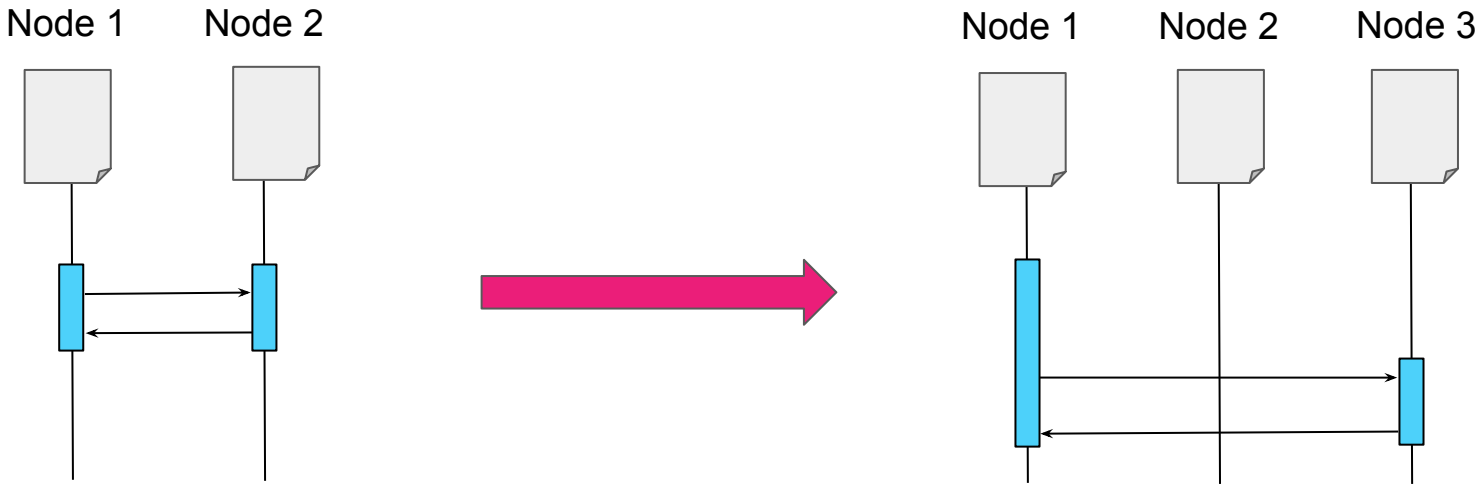
Adding nodes: choose one partition

The replication group for partition <P> is changed from {1, 2} old distribution to {1, 3} new distribution.



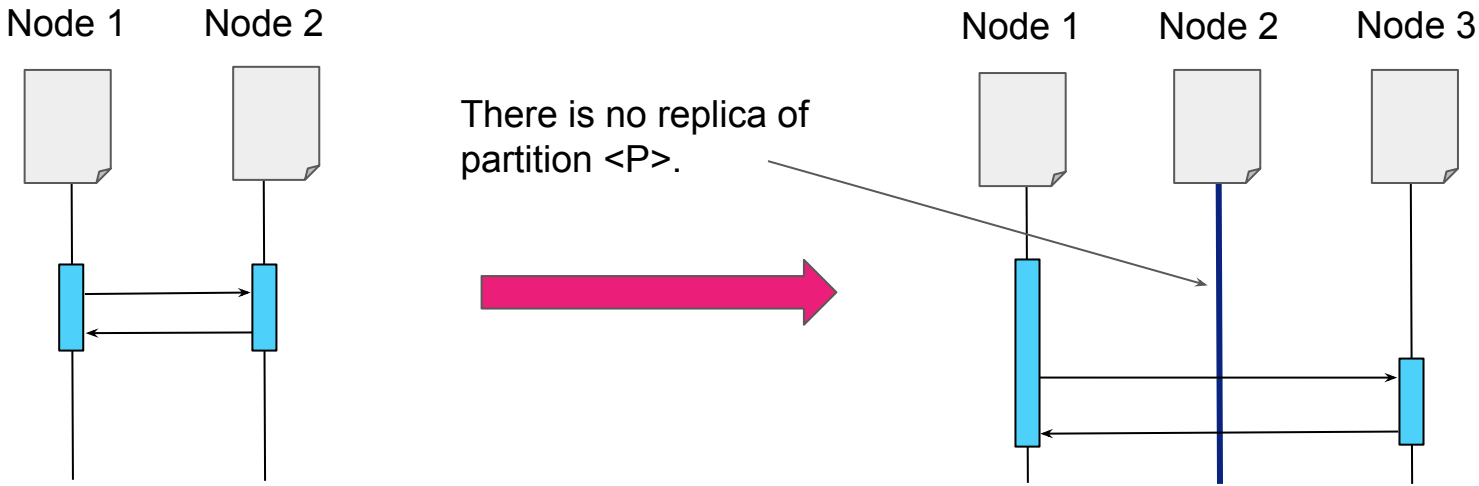
Adding nodes: changing in replication groups

The replication group for partition <P> is changed from {1, 2} old distribution to {1, 3} new distribution.



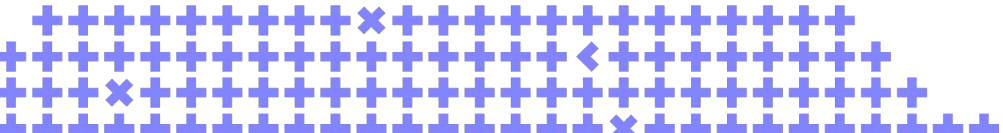
Adding nodes: changing in replication groups

The replication group for partition <P> is changed from {1, 2} old distribution to {1, 3} new distribution.

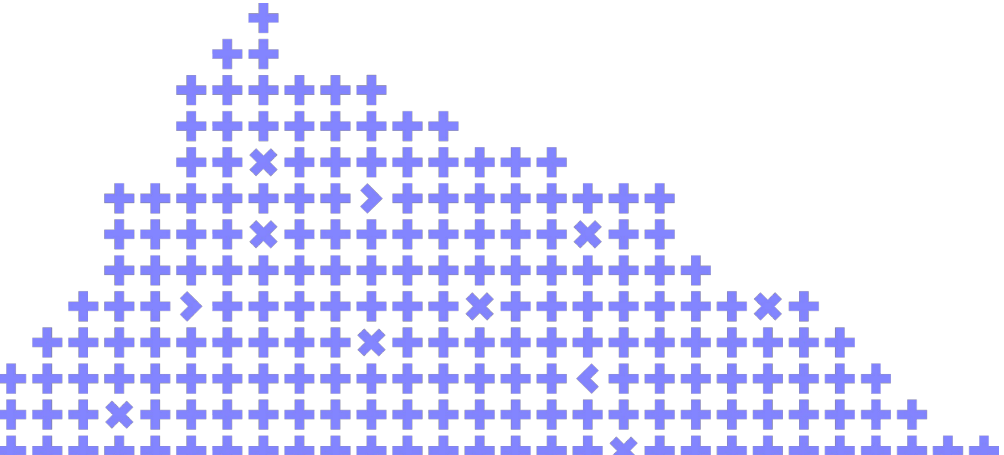


Redistribution steps

- Recalculate a distribution in a new topology
- Determine partitions whose replicas have to be moved
- The partitions are changing their replication node sets to an intermediate distribution
- The replicas need to be copied from one node to another
- After the replicas are moved the cluster switches to the new distribution

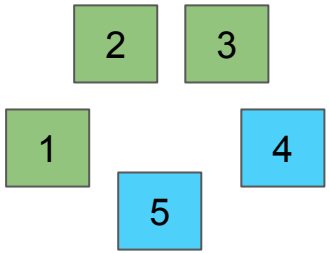


What happens if all partition replicas should be moved to new nodes during redistribution?



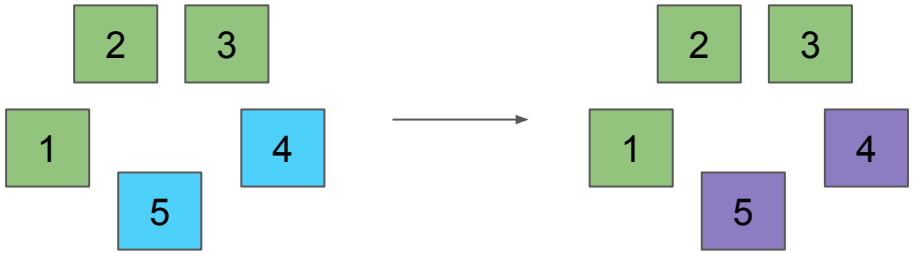
Modifying the distribution

The cluster has intent to modify the distribution {1, 2, 3} to a new one {4, 5}



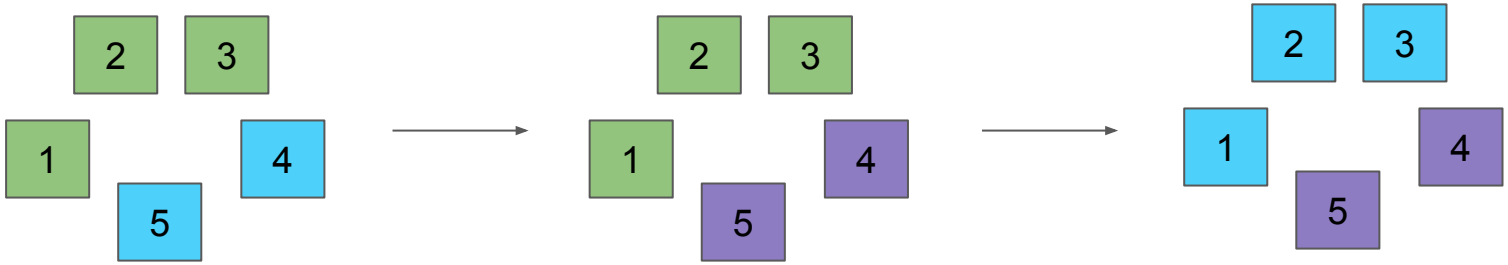
Modifying the distribution

The cluster has two different distributions, the previous distribution {1, 2, 3} and the new one {4, 5}



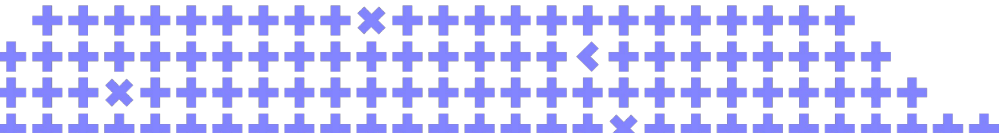
Modifying the distribution

The new distribution {4, 5} fully replaces the previous one



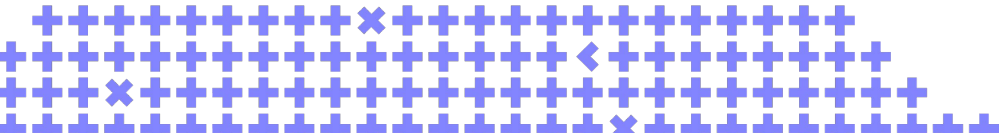
RAFT

- RAFT is a consensus algorithm based on a log replication
- The protocol process writes to one node (leader) and reads from any node
- The log replication acts by moving of log records
- A snapshot transfer is also supported by the algorithm
- The algorithm allows changing of a replication group (quorum)



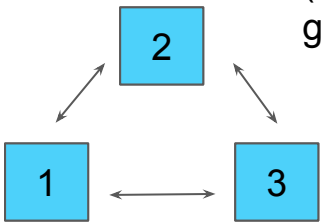
RAFT consensus

Makes the protocol reliable in case of split brain



RAFT consensus

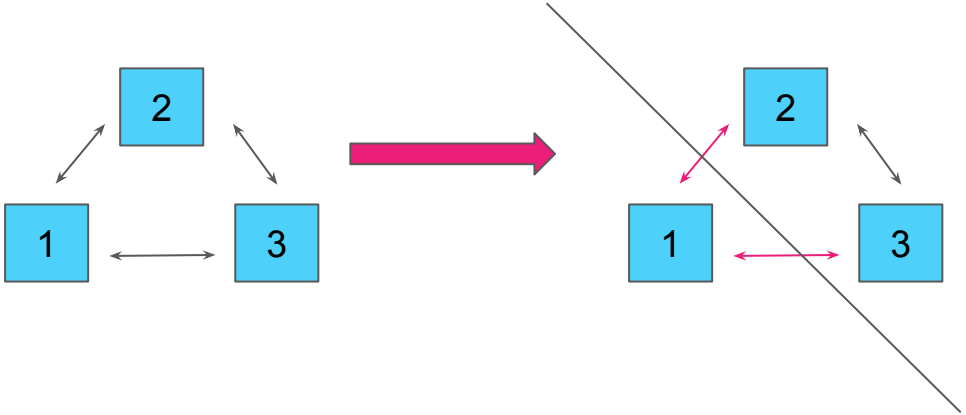
Makes the protocol reliable in case of split brain



Each of the three nodes contains a partition replica (the nodes are a replication group)

RAFT consensus

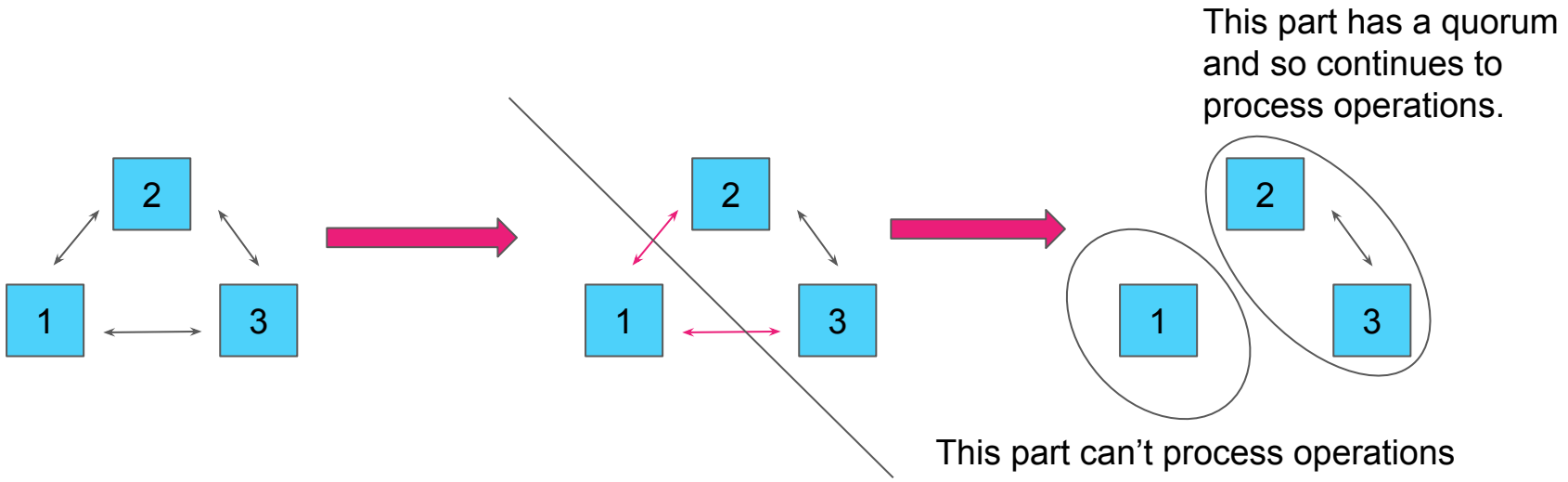
Makes the protocol reliable in case of split brain.



The connection with node 1 is broken.

RAFT consensus

Makes the protocol reliable in case of split brain



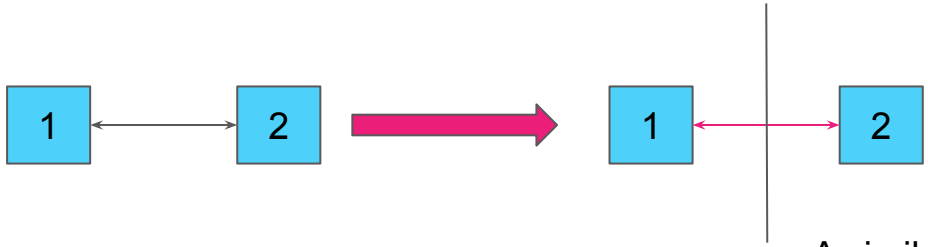
RAFT consensus

Makes the protocol reliable but has limitations.

Two nodes contain the same partition replica.

RAFT consensus

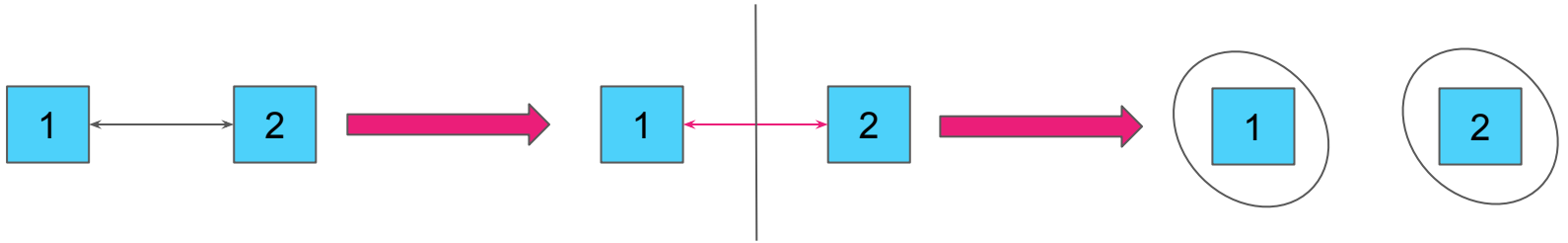
Makes the protocol reliable but has limitations



A similar network issue happens.

RAFT consensus

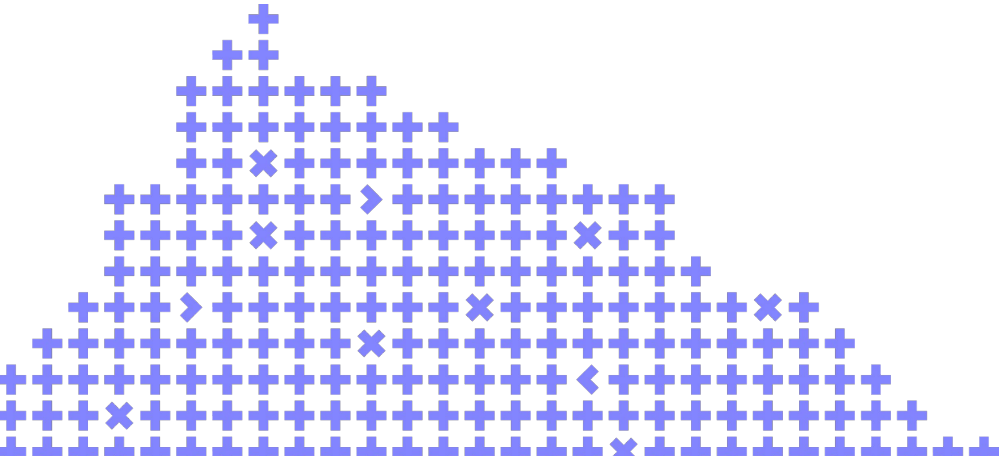
Makes the protocol reliable but has limitations



Neither part can process operations

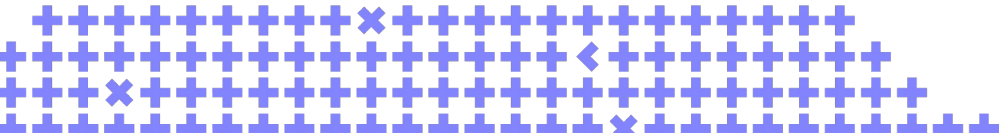
The cluster is unavailable,
but the data is not lost.

Conclusion



Conclusion

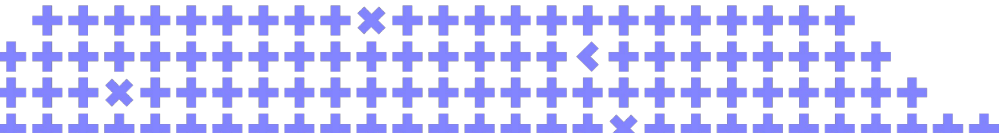
- ➜ Both replication and rebalance are required to use for data movement in any distributed database.



Conclusion

- Both replication and rebalance are required to use for data movement in any distributed database.

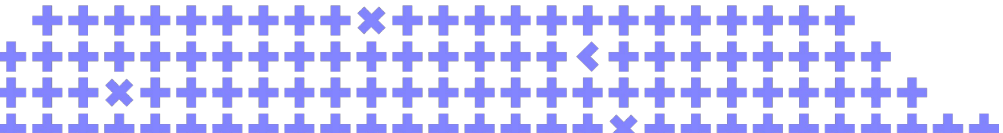
➜ Using log records for replication allows using replication for rebalance purpose.



Conclusion

- Both replication and rebalance are required to use for data movement in any distributed database.
- Using log records for replication allows using replication for rebalance purpose.

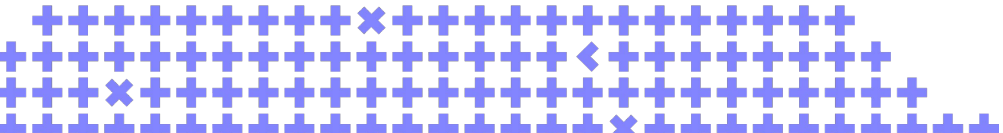
➜ Replication does not care about concurrent data updates.



Conclusion

- Both replication and rebalance are required to use for data movement in any distributed database.
- Using log records for replication allows using replication for rebalance purpose.
- Replication does not care about concurrent data updates.

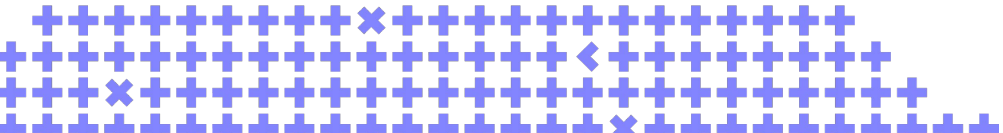
➜ Using a log for replication is better than using snapshots for replication.



Conclusion

- Both replication and rebalance are required to use for data movement in any distributed database.
- Using log records for replication allows using replication for rebalance purpose.
- Replication does not care about concurrent data updates.
- Using a log for replication is better than using snapshots for replication.

➜ RAFT is a reliable algorithm for copying logs, but it has limitations.



Leave your feedback!

You can rate the talk and give feedback on what you've liked or what could be improved

Vladislav Pyatkov
vpyatkov@gridgain.com

https://bit.ly/3W828Xa

Co-organizer

Yandex